A Logical Form Parser for Correction and Consistency Checking of LF resources

Rodolfo Delmonte, Agata Rotondi

Department of Linguistic Studies and Comparative Cultures and Department of Computer Science
Ca Foscari University - 30123 VENEZIA (Italy)
delmont@unive.it

Abstract. In this paper we present ongoing work for the correction of Extended WordNet (XWN), the most extended freely downloadable resource of Logical Forms (LFs) – by the Human Language Technology Research Institute (HLTRI) of University of Texas at Dallas (UTD). In a previous paper we reported on type and number of errors detected in the 140,000 entries of the resource, which amounted to some 30%. This didn't include problems related to inconsistencies from disconnected variables which were not computable at the time. We now created an LF parser that parses each entry after appropriate transformations. The parser has been created to count the number of disconnected variables, be they object variables or predicate event variables: the result is 56% of LFs containing some disconnected variable. We devised two procedures for correction: one lexical and the other structural which eventually allowed a dramatic reduction: the final count is now 24%. Additional work has been carried out to improve the general consistency by manual intervention on "inconsistent" outputs signaled by the parser and has reduce the number of errors to a reasonable percentage for such a resource, that is less that 15%.

1 Introduction

This paper presents work carried out to parse and correct LF resources and in particular XWN or Extended WordNet (see Mihalcea et al. 2001), a freely downloadable resource containing a mapping into Logical Forms (LFs) of WordNet glosses. We started to produce a parser of LFs after working at individuating errors in XWN (see XXX). After we discovered that some 30% of all entries needed some corrections, we decided to continue work for a number of reasons, some of which positive some others negative that will be discussed below.

As to negative question, the first regards the way in which syntactic structure has been produced. Current parsers mostly produce surface dependency/constituency structure with good enough approximation, but which is of no use for the mapping into LFs seeing that deep relations are missing. We are here referring to two types of parsers, Charniak's constituency-like parsers and dependency-like parsers. Deep parsers are only a few and the accuracy of their performance is insufficient for a mapping into LFs seeing that LFs require fully consistent representations in order to preserve the semantics (but see below).

Now, there is a number of applications that produce LFs directly from syntactic representations, but their performance is unsatisfactory for the reasons explained in the previous paragraph. We tried LFToolkit (Rathod, Hobbs, 2005) which maps into LF directly from the output of Charniak's parser - more on this below. Worthwhile mentioning is also Cahill et al. (2007) attempt at transforming a portion of WSJ into LFs by means of a conversion of PennTreebank II augmented syntactic representation into complete F-structures. The authors claim an F-measure of 97% over the 96% of sentences converted, which is certainly a success. However, WSJ sentences are in no way comparable to glosses and their online parser does not allow the creation of LF[1]. Since LFs cannot be produced fully automatically and need a lot of manual additional work, we thought it reasonable to try and use existing LF resources such as ILF (Intermediate Logical Forms) - see Agerri & Peñas, 2010, XWN and others. We believe it is always worthwhile correcting these resources, wherever possible. WordNet glosses are definitions, meaning paraphrases and declarative descriptions associated to synsets of WN, which is what makes them highly valuable for semantically heavy tasks such as Q/A, WSD, and Text Understanding in general.

Coming now to the actual resources, LF mapping from PT (Penn Treebank)-like constituency-based syntactic structures are - in our opinion - a lot more error prone than those derived from dependency structure (see Agirre & Peñas, 2010). This is due to the fact that PT-like structures are more difficult to map due to the nature of constituency structure, which is more functionally based than semantically oriented, when compared to dependency structures. Syntactic constituency in PT as well as the one produced by Charniak's parser, associates main constituency nodes to functional heads like auxiliaries, complementizers, subordinating conjunctions,

[1] http://lfg-demo.computing.dcu.ie/lfgparser.html

relative and interrogative pronouns, modals, verb particles. Nominal heads are usually lumped together with their determiners and modifiers, be they other nouns or adjectives. For this reason, using a dependency structure in which semantic heads are separated from functional ones can be and is – as ILF clearly shows, but see below – highly beneficial for a safer mapping into LF. In this sense, ILF being mapped from dependency structures is much closer to semantic content than XWN - more on ILF below.

Both XWN and ILF have been mapped without the help of additional resources such as lexica and anaphora resolution algorithm, which were in fact necessary, as will be shown below. However, the net result is the absence of free ungrounded variables in ILF: on the contrary in XWN, the presence of ungrounded variables is the rule, as will be shown below. This is partly due to the lack of any one to one correspondence between constituency structure and LF as encoded in the mapping algorithm.

The paper is organized as follows: in section 2 we will review different types of Logical Forms and try to evaluate the contribution of each representation; in section 3 we concentrate on XWN, WNE and ILF and individuate their strengths and weaknesses; section four is dedicated to presenting the parser itself; then we end with some evaluation, conclusion and future work.

2 Logical Forms, but what kind?

In XWN, WNE and ILF the mapping to LFs has been done semi-automatically with manual checking of the majority of syntactic constituency structures. However, the actual mapping process has not been subsequently checked nor evaluated (but see Vasile, 2004).

What kind of LF are we referring to? The LF we are referring to is a flat unscoped first-order logic (FOL) well-formed formula representing the "meaning" of a sentence. It has been restricted to a conjunction of predicates which in turn contain arguments that have been hampered from being themselves recursive. Possible arguments of predicates can be event variables and argument variables, the latter being also called object variables, referring to entities, properties and attributes.

However, not all work on Logical Form would look the same and there are lots of different ways of computing and building them. In XWN, for instance, there is no attempt at covering all if not most of what is commonly regarded as semantically related problems that might as such be represented in a LF. Here below we show some valuable attempt at including some of the semantics in the LF from the contents of the book by (Bos & Delmonte, 2008) for the workshop of ACL Sigsem held in Venice.

The first LF we show is the one used by J.Bos to represent DRS. As can be seen below, variables introduced in the representation are all of the same kind, the prefix always being X. What changes is the number that follows the X. As a result there is no distinct event variable, with an E prefix. The text is one of the Shared Task of the workshop and we take these two sentences (ibid., 283):

Sent.1 Cervical cancer is caused by a virus.

Sent.2 That has been known for some time and it has led to a vaccine that seems to prevent it.

```
|x0 x1 x2     |      | x3 x4 x5  |                                  |event(x10)
| thing(x0)          | cancer(x3)          |x6 x7 |               |event(x8)
| neuter(x1)         | cervical(x3)        |know(x6)              |agent(x8,x1)
| neuter(x2)         | cause(x4)           |time(x7)              |agent(x10,x9)
|_____________       | virus(x5)           |event(x6)             |theme(x10,x11)
                     | event(x4)           |theme(x6,x0)          |to(x8,x9)
                     | theme(x4,x3)        |for(x6,x7)            |_____________
                     | by(x4,x5)           |_____________
                     |_____________                               |x11:x12
                                           |x8  x9  x10  x11      |prevent(x12)
                                           |lead(x8)              |event(x12)
                                           |vaccine(x9)           |agent(x12,x9)
                                           |seem(x10)             |theme(x12,x2)
                                           |proposition(x11
```

In this LF, events have been reified and appear as functions heading their variable. Also semantic/thematic roles have been reified, and head the variables of both argument and event. This choice multiplies the number of logical objects, but simplifies the matching. Another way of rendering the LF of sentence 1, could have been

```
cause(e4,x5,x3),cancer(x3),cervical(x3),by(e4,x5),virus(x5)
```

A slightly similar approach has been taken by (Clark et al., 2008) with a system that also comprises a parser and a logical form generator[2]. Their example is shown below, where variable are indicated by underscored _X:

```
Sent.3 "A soldier was killed in a gun battle."
(DECL ((VAR _X1 "a" "soldier")
(VAR _X2 "a" "battle" (NN "gun" "battle")))
(S (PAST) NIL "kill" _X1 (PP "in" _X2)))
```

This mixed syntactic structure is then used to generate "ground logical assertions of the form r(x,y), containing Skolem instances, by recursively applying a set of syntactic rewrite rules to it. Verbs are reified as individuals, Davidsonian-style."(ibid., 48; but see also Davidson, 1967;1980):

```
object(kill01,soldier01)
in(kill01,battle01)
modifier(battle01,gun01)
```

As the authors comment, predicates used in this representation are just syntactic relations of the type SUBJect_of, OBJect_of, and MODifier_of and all prepositions, which typically take two variables related to the individuals they are bound to. In particular, in this representation Skolem instances are associated with its corresponding input word. Syntactic relations represent deep relations: the surface subject of the passive sentence Sent.3 is turned into an OBJect.
Another richer way of representing meaning in LF is proposed by Delmonte in Bos & Delmonte 2008:291, for the sentence,

```
Sent.4 John went into a restaurant
wff(situation,
    wff(go,
     < entity : sn4 : wff(isa, sn4, John) >,
     < indefinite : sn5 : wff(isa, sn5, restaurant) >,
     < event : f1 : wff(and, wff(isa, f1, ev),
       wff(time, f1, < definite : t2 :
      wff(and, wff(isa, t2, tloc), wff(pres, t2)) >)) >))
```

where we see that two semantic elements appear in the representation, DEFINITENESS, and TENSE which is associated to a Reference Time location variable, T2. As will be discussed below, Reference Time and Definiteness are two important semantic features and are introduced also in other LF representations.

3 Previous Work related to XWN

There is a certain amount of additional work carried out on XWN that we want to review briefly here below. Apart from XWN by UTD people, the other effort to translate WN Glosses into Logical Form is by people at USC/ISI California, in 2006. Their results are also available on the web and freely downloadable at ISI, 2007. As the comment on the related webpage clearly states, "The following additional "standoff" files providing further semantic information to supplement the WordNet 3.0 release." The file contains LFs in XML format for most of the glosses "except where generation failed" as the comment clearly warns out. The authors made a subset of the core WordNet including 2800 noun senses in plain text format, in 2007. As the comments on the website say, "these are generally of higher quality than those contained in the file below for all glosses." We find these representations in eventuality notation too cluttered with additional event variables, which makes the LF entry too heavy to read, as can be seen in the example of the entry BUTTER included below. These files can be downloaded at http1;http2. The conversion process of WN glosses proceeds by parsing with Charniak parser and the result is converted into a logical syntax by a system called LFToolkit (Rathod & Hobbs, 2005). Each lexical semantic head is transformed into logical fragments involving variables. For instance "John works" - commented in detail below - is translated into John(x1) & work(e,x2) & present(e), where e is a working event. Object variables are differentiated at first (x1 and x2), and then a rule which recognizes “John” as the subject of “works”

[2] In the authors' words, the LF is "a semi-formal structure between a parse and full logic, loosely based on Rathod & Hobbs, 2005. The LF is a simplified and normalized tree structure with logic-type elements, generated by rules parallel to the grammar rules, that contains variables for noun phrases and additional expressions for other sentence constituents. Some disambiguation decisions are performed at this stage (e.g., structural, part of speech), while others are deferred (e.g., word senses, semantic roles), and there is no explicit quantifier scoping. "

sets x1 and x2 equal to each other. This works for the majority of English syntactic constructions. As the authors themselves comment, whenever there was a failure by the LFToolkit, it happened as a result of a bad parse, due to the presence of constructions for which no rule in LFToolkit had been written[3].

I will show here below the entry for BUTTER as it has been transformed by the two systems. The first representation is the one produced at USC/ISI and the second one is the one by XWN in (Moldovan & Rus, 2001). In fact both representations are in XML format, but for easiness of reading we eliminate angled brackets:

```
example (1)
entry word="butter#n#1" status="partial"
gloss = an edible emulsion of fat globules made by churning milk or cream;
for cooking and table use
butter#n#1'(e0,x0)    ->    edible'(e9,x1)    +    emulsion#n#1'(e1,x1)    +
of'(e6,x1,x12)      +      fat#n#1'(e15,x17)+nn'(e14,      x17,x12)      +
globule#n#1'(e10,x12)  +  dset(s5,x12,e10+e14)  +  make#v#15'(e2,x4,x3,x2)  +
by'(e3,x5,e7)   +   churn#v#1'(e7,x10,x14)   +   milk/cream#n#2'(e11,x14)   +
for'(e4,x6, x11) + cooking'(e12,x16) + table'(e13, x15)
milk'(e11, x14) -> milk/cream#n#2'(e,x14)
cream#n#2'(e11, x14) -> milk/cream#n#2'(e,x14)

example (2)
butter:NN(x1)  -->  edible_JJ(x1)  emulsion:NN(x1)  of:IN(x1,x2)  fat:JJ(x2)
globule:NN(x2)     make:VB(e1,x9,x1)     by:IN(e1,e2)     churn:VB(e2,x2,x5)
milk:NN(x3) or:CC(x5,x3,x4) cream:NN(x4)
```

As can be seen in example(1), all lexical items are treated as predicates and have an event variable starting with E, associated to them. Event variables are typically unbound and should be quantified over. They are associated to object variables which start with X. In some cases, when a DSET is asserted, event variables are connected explicitly to their object variable, as in the nominal compound "FAT GLOBULES". Also this LF representation, which is classified as PARTIAL, contains a lot of unbound or ungrounded variables, as for instance in the case of MAKE(e2,x4,x3,x2) in example (1), where none of the object variables have an individual ground object linked to them.

Example(2) is much simpler and shorter. In this case, the LF representation produced has a better output. However, if we look at the representation associated to MAKE, we notice that only has three variables, one of which is the event variable, e1, and the remaining two are argument variables - x9, x1. Whereas x1 is properly bound to the entry BUTTER, the second variable is unbound. We can also notice that the first representation treats MAKE as a 3-place predicate, as in the sentence "John made the butter smooth by...". On the contrary, the second representation only has two argument variables: this could be justified by the use of MAKE in a participial structure, with a different meaning though. The meaning in this case is obtained by omitting the secondary predication. It is just a simple transitive structure in a passivized form. More on this topic below.

The problem related to these examples are typical problems of mapping from surface syntactic structures to Logical Forms, and we have tried to overcome them by building an LF parser that checks for consistency. Here the term consistency is referred solely to the existence of free unbound or ungrounded variables in a given LF representation. This fact does not allow relations indicated by predicates to be associated to arguments and modifiers, which are thus disconnected. In this way, the formula is useless and meaningless. Variables associated to predicates needs to be equated with those of the arguments of the predicate in order to acquire semantic consistency. From our analysis, 54.05% of all LFs contained in XWN suffer from that problem. In particular, they constitute 71,658 over 132,587 where we found the following data:

categories	Dis.Vars	Tot.LFs	%
Adverbs	487	3982	12.23
Adjectives	8886	20317	43.74
Verbs	9751	14454	67.46
Nouns	52672	94028	56.00
Total	71,658	132,587	54.05

Table 1. Errors detected by the parser

[3] "In these cases, the constituents are translated into logic, so that no information is lost; what is lost is the equalities between variables that provides the connections between the constituents. For instance, in the “John works” example, we would know that there was someone named John and that somebody works, but we would not know that they were the same person. Altogether 98.1% of the 5,000 core glosses were translated into correct axioms (59.4%) or axioms that had all the propositional content but were disconnected in this way (38.7%). The remaining 1.9% of these glosses had bizarrely wrong parses due to noun-adjective ambiguities or to complex conjunction ambiguities."(ibid.,49)

Here percent values refer to errors found by the parser.

4 The LF Parser

The parser takes as input two files, one containing the list of logical forms as they have been listed in XWN for the different categories - verb, noun, adjective, adverb; and another file containing the synset offset followed by the synset. Each synset is associated to a gloss that contains one or more definitions, comments or examples: this is what has been transformed into Logical Forms in XWN. So the parser knows that there may be one of more LFs to associated to the same synset offset index. Now, each Logical Form will necessarily start with the same lemma which corresponds to the first lemma in the synset: for instance, the entry 00002931 of the ADJ dataset corresponding to the synset "abducent, abducting", has the associated gloss "especially of muscles; drawing away from the midline of the body or from and adjacent part". This gloss is transformed into two LFs, respectively,

```
abducent:JJ(x1)-> draw_away:VB(e1,x1,x5) from:IN(e1,x2) midline:NN(x2) of:IN(x2,x3)
body:NN(x3) from:IN(e1,x4) adjacent:JJ(x4) part:NN(x4)

abducent:JJ(x1) -> especially:RB(x1) of:IN(x1,x2) muscle:NN(x2)
```

These have then been turned into the Prolog compliant corresponding structures below:

```
lf(abducent_JJ(x1),[draw_away_VB(e1,x1,x5),from_IN(e1,x2),midline_NN(x2),
of_IN(x2,x3),body_NN(x3),from_IN(e1,x4),adjacent_JJ(x4),part_NN(x4)]).

lf(abducent_JJ(x1),[especially_RB(x1),of_IN(x1,x2),muscle_NN(x2)]).
```

The parser takes the synset offset associated to the current synset and the first LF in the current list. Then it matches the first lemma in the synset with the lemma heading the LF. After correcting the LF, the parser checks the rest of the list to see whether there is another occurence of the current lemma and in that case it keeps the same offset index, otherwise it passes the rest of the synset-offset list. As will be discussed in detail in a section below, this version of the algorithm works fine only for a part of WordNet, proper names behave differently and the algorithm had to be modified to cope with them. The output of the algorithm is a conjunction of the information contained in the two files, as follows:

```
synset(300002931,abducent_JJ(x1),[abducent,abducting])-[draw_away_VB-[e1,x1],
from_IN-[e1,x2],midline_NN(x2),of_IN(x2,x3),body_NN(x3),from_IN-[e1,x4],
adjacent_JJ(x4),part_NN(x4)]

synset(300002931,abducent_JJ(x1),[abducent,abducting])-[especially_RB(x1),
of_IN(x1,x2),muscle_NN(x2)]
```

There are at least three different ways of conceiving the relation intervening between variables in a flat unscoped LF. As discussed above, the simplest way would be that of considering all variables free and each one different from the others, and then at the end, specifying those variables that have to be regarded equal by additional equations. A second way, is to regard all variables equal, and then specifying the ones that have to be regarded different - and this is what has been done in Robust Minimal Recursive Semantics, (RMRS) (Copestake, 2009). In both these two ways, however, variables need to be precisely bound as required, which is not what actually happens. We report one of the examples from that presentation, for the sentence "Some big angry dogs bark loudly", where we see that a scoped LF is used to convey the role of quantifier "some":

```
example (3)
some_q(x4, big_a_1(e8,x4) ∧ angry_a_1(e9, x4) ∧ dog_n_1(x4), bark_v_1(e2,x4) ∧
loud_a_1(e10,e2))
```

Notice that in this LF representation, every attribute and modifier has a separate event variable name. This is remarkably different from what has been done in XWN.

The third way, which is more consistent with what has been done in the XWN LF representation, is to consider variable equalities to indicate relations of some kind: in particular, any modification relation is indicated by variable equality; the same applies to argument relations. Another important topic regards the way in which optional or omitted arguments should be treated in the LF representation. As discussed in (Copestake, 2009) LFs

for predicates should consider deep structure information rather than simply surface structure, and in case an argument is missing - because it has been omitted in a passive structure or simply because optional - this should be signaled appropriately by marking the corresponding slot with U (for unexpressed). These are some of the problems that we will try to tackle in our parser.

3.1 The architecture of the LF parser

The LF parser is organized in a pipeline:
A. the first module tries to match variables in predicates with their object counterpart.
B. the second module does the opposite: it tries to match variables in object formulas with their predicate counterparts.
This has to account for a number of different logical structures. The most common one is the one accounting for predicate argument structures governed by verbs, as in,
`- buy_VB(e1,x2,x1)`
where e1 is a generic event variable which might or might not have higher level binders, or meta level formula (see below) associated to it; x2 is by slot convention a variable associated to the complement (in this case an object); and x1 is again by slot convention the variable associated to the subject, treated as external argument. A second structure is the one associated to prepositions and other similar two place relation markers as comparative conjunctions or even subordinators and relative pronouns:
`- of_IN(x2,x3) - than_IN(x4,x5) etc.`
where x(number) variables bind objects, and prepositions - when they introduce gerundives - and (subordinating) conjunctions treated as relation markers:
`- by_IN(e1,e2) - since_IN(e4,e5)`
All relation markers only contain relational variables and no event or object variable of their own.
Object formulae include simple one place predication with just one variable associated to an entity, a property or an attribute, as in
`- dog_NN(x2)`
XWN uses the same specification also for modifiers like adjectives and adverbials:
`- angry_JJ(x2), fast_RB(e2)`
but this, on the basis of what we have commented above, needs some reorganization. Modifiers are then supplemented by an additional variable of their own that accounts for the role of predicate they fulfill. This will allow to differentiate cases in which the same adjectival word - say "red" - may play the role of predicate in a copulative construction which has to be differentiated from the role of attribute in a nominal compound, as in (4b) "The red hat was stiff" vs. (4a)"The stiff hat was red". The two sentences could be differentiated as follows, where x1 is associated to the subject of predication, "hat" and the predication itself is constituted by a different property identified by variable e3. The attribute is associated to the nominal head object variable x1 and is specified with event variable e2, assuming in this way that the property of being “stiff” is independent of the property of being “red”, but they are both associated to the entity X1:

```
example(4a)
be_VB(e1,x1), hat_NN(x1), stiff_JJ(e2,x1), red_JJ(e3,e1)
example(4b)
be_VB(e1,x1), hat_NN(x1), red_JJ(e2,x1), stiff_JJ(e3,e1)
```

There are then mixed formulae which include both event and object variables, as in,
`- by_means_of_IN(e1,x5) = (buying) by means of`
`- consider_VB(e3,e1,x2,x1), silly_JJ(e1,x2) = consider (your dog) silly`

where x2 is the variable associated to "dog" and e1 is included in the argument list of the verb. This is what differentiated real copulative verbs like "be", and transitive verbs like CONSIDER which have secondary predication as argument. As anticipated above, there are also meta-level formulae and they are of two types:
- coordinating conjunctions
- complex nominal compound
The first refers to coordinating conjunctions, which allow to refer to sets of objects or predicates. The latter are separately specified: here, rather than duplicating the noun governors, the meta abstract coordinating conjunction is used, as shown below. Coordination may interest both event and object variables, as follows:
`- and_CC(x6,x1,x2,x3)`
`- decide_VB(e4,e5,x6), leave_VB(e5,x6)`
for the sentence, "Frank, John, and the dog decided to leave".
A similar formula would be used in case event variable should be coordinated as in,

- or_CC(e8,e1,e2,e3)

where e8 would be the variable associated to the coordinate structure.

The other meta level formula is associated to the function NN introduced in the XWN following Jerry Hobb's suggestions. In this case, the only possible set of variables is the one associated to objects or nominals indicating properties of the head, usually the last variable in the set. As in one below for "Samsung's profits" where x4 and x6 are bound to single entries,

- NN(x7,x4,x6)
- Samsung_NN(x4), profit_NN(x6)

and the variable x7 would be used by the event predicate that governs the nominal compound,

- rise_VB(e2,x7)

In order to allow for smooth matching procedures, all meta-level formulae are turned into their simple binary level by reification. Reification is also used for negation as shown below:

- coordinations are turned into one place predicates,

- and_CC(x6,x1,x2,x3) --> and_CC(xc), coord(xc,x6), coord(xc,x1), coord(xc,x2), coord(xc,x3)

- negations are reified:

- not_RB(e2) --> neg(xn,e2),not(xn)

In fact, all negation formulae had to be corrected before starting to check for their consistency. We eliminated all DO auxiliaries and associated the negation predicate directly to the main verb variable, using regular expressions. In this way we got a double result: unwanted auxiliary information was eliminated and the negation operator is now correctly associated to the main verb meaning. A similar change had to be introduced for all cases of wrong treatment of the amalgam CANNOT, which in XWN is introduced directly without a decomposition, and in many cases is wrongly tagged as noun. So we produced the following change again by regular expression, where we deleted the variable associated to the wrong tagging and substitute it with the right one.

3.2 Correcting Logical Forms

We envisage two types of corrections: one induced by lexical information and another by structural information. The first correction is addressed to all those predicates that contain a dummy variable for an argument which does not exist in reality. In fact, here we are referring to verbs belonging to classes like unergative verbs, unaccusative verbs, weather verb, impersonal verbs, but also to verbs which can be intransitivized, ergativized. That is verbs which induce intransitive structures, either by raising the object to subject position and eliminating the deep subject; or cases of verbs which allow the object to be left unexpressed, that is something which can be quantified over by an existential quantifier. Always on the basis of lexical information, we check for intransitivized and passivized TRANSITIVE verbs, which constitute by far the majority of cases. In particular, in case a passivized past participle is being used, this is usually accompanied by the omission of the deep subject. As to the structural corrections, we have been filtering wrong structures by a procedures that allows the correction module to select only those parts of the formula which need to be modified. In order to extract information related to wrong and inconsistent LFs, the parser collects variables related to object formula separately from those related to predicate formula. Then it does a simple intersection. The set of intersecting variables is then used to verify whether there are ungrounded variables.

We used the two procedures in a sequence – at first we found ungrounded variables and then looked for predicates with unneeded variables that coincided with the ones found in the previous procedures – and eliminated them. The results are remarkable: we managed to eliminate some 32%, that is almost half of previous 72% of all disconnected variables. In particular, they now constitute 31,583 over 132,587 – 23.82%.

categories	Dis.Vars	Tot.LFs	% after	% before
Adverbs	250	3982	6.28	12.23
Adjectives	1599	20317	7.87	43.74
Verbs	823	14454	5.69	67.46
Nouns	28911	94028	30.75	56.00
Total	31,583	132,587	23.82	54.05

Table 2. Errors after parser correction

As said above, XWN introduces a first event variable e1 or sometimes e0, which should be quantified over and are left unbound. Also a first object related variable is always associated to nouns and adjectives, and it is X1. These are not considered in the intersection and are removed from the set. Here below some examples of inconsistent formula for ABLE:

gloss: having the necessary means or skill or know-how or authority to do something

```
able:JJ(x1) -> have:VB(e1, x1, x8) necessary:JJ(x8) means:NN(x2)
skill:NN(x3) know-how:NN(x4) or:CC(x8, x2, x3, x4, x5) authority:NN(x5)
to:IN(x8, e2) do:VB(e2, x8, x6) something:NN(x6)
```

where DO has x8 as Subject variable, which should be x1, that is the person that is ABLE, SUBJect of the predication of HAVE and also head of the adjective modifier. More errors are contained in the formula, where necessary(x8) should be necessary(x2), seeing that it only modifies "means". The following case is an inconsistency caused by various errors: "dependent_on" is associated to an unground variable "x4" and not "x1"; the same applies to "relative" which is associated to "x2", rather than "x1":

gloss: not dependent on or conditioned by or relative to anything else

```
independent:JJ(x1) -> not:RB(e2) dependent_on:JJ(x4) condition:VB(e1, x5,
x1) by:IN(e1, x5) or:CC(e2, e1) relative:JJ(x2) to:IN(e2, x2)
anything:NN(x2) else:JJ(x2)
```

here below, we show what the correct LF would be like after introducing predicate variables in each adjective modifier, changed ungrounded variable associated to obligatory argument into an undefined U variable:

```
independent_JJ(x1) -> not_RB(e3), dependent_on_JJ(e1,u,x1), or_CC(e3,e1,e2),
condition_VB(e2,x2,x1), by_IN(x2,u), or_CC(e4,e3,e5),
relative_to_JJ(e5,x3,x1), anything_NN(x3), else_JJ(x3)
```

where DEPENDENT_ON has become a complex phrasal predicat. In its formula, the preposition ON requires an additional variable, ungrounded, though; and RELATIVE has also become a phrasal adjective with preposition and as such in need of an additional argument variable. To this end, we turned both adjectives into two place predicates with an event variable to indicate that there is a dummy BE verb implicit in the gloss.

As said above, the parser at first measures intersection: in case no intersection intervenes then a flag is written on the output file and used by the correction module. In the following case, for instance, after deleting x1 and e1, the intersection is empty.

```
INPUT
lf(approved_JJ(x1),[generally_RB(e1),especially_RB(e1),officially_RB(e1),ju
dge_VB(e1,x5,x1),acceptable_JJ(x3),satisfactory_JJ(x3)]).

OUTPUT
approved_JJ(x1)  6  [x5,x3] no intersection
lf(approved_JJ(x1),[generally_RB(e2,e1),especially_RB(e3,e1),officially_RB(
e4,e1),judge_VB(e1,e5,u,x1),acceptable_JJ(e6,e5),satisfactory_JJ(e7,e5)]).
```

where we see that APPROVED has an LF formula made of 6 elements, that the intersection of the relevant variables is empty, and that there are two variables in particular which have no correspondence in object formula. The parser will inspect the formulas one by one, and eventually will equate x5 with x3, thus making the whole LF consistent. The equation decision is determined by the fact that: x5 is contained in a predicate formula, which also contains x1, and that x3 are both contained in object formula. In addition they both FOLLOW the predicate, this being a clear indication - in English at least - that they constitute a COMPLEMENT to the predicate itself. The modified and corrected formula contains also additional event variables for attributes predicated to x3 and an U for unexpressed argument variables.

In particular, the system found 110 cases of inconsistencies in the ADVERBS file of XWN, 1154 cases of inconsistencies in the ADJECTIVES file, 2054 in the VERBS file and 5276 in the NOUNS file. Overall, 8594 cases that we addressed by the correction module. The parser managed to correct half of them in a first run. Then more rules have been devised to correct the rest of the errors. These rules have then been used to check and correct most of the remaining LFs.

In fact, as a whole, we managed to correct many more entries, thanks to the fact that the parser simply got stucked whenever the LF entry was not computable, i.e. none of the variables matched either x1 or e1. We also corrected all negation operators and some of the conjunctions which were not tagged consistently, as for instance THEN, which was tagged as RB (adverbial) most of the time, and only sometimes as IN.

The evaluation of the corrections produced manually and automatically is made directly by the parser itself. The output of the parser, in the correction mode, is a file containing all LFs which have some inconsistency. Corrections have been carried out specifically on the output of the parser. Evaluation in this case is computed accordingly on the basis of number of mistakes found by the same parser.

3.3 The case of Proper Names in WN

The parser works smoothly with three files, the ones containing Adverbs, Adjectives, and Verbs, but when the file containing Nouns is started problems arise which eventually obliged me to modify the algorithm. WordNet contains some 18K capital letter initial lemmata which can be computed as proper names or named entities, i.e. person names, organization names, famous events names, institution names, location names. Contrary to expectations, the description of these entries in the database resembles the one used for common nouns, which as we know are used to denote classes of individuals, whereas proper names would rather be used to individuate uniquely a single referent in the world - they are rigid designators according to Kripke[4]. In fact, this is only partially true, seeing that a proper name made by the compound of first name and surname today can be regarded ambiguous and can refer to different referents in the world. Now, let's consider synsets: synsets are a collection or set of synonym lemmata which may constitute a single concept in a specific language. Lexica of different languages may vary a lot on this and a synset made of a plurality of referent words for the same concepts, translated in another language may turn up to be uniquely denoted by one single lemma. The other dimension of synsets is their polysemous nature: in this case, the same lemma - in case the synset is a singleton - or the first lemma of a set, may be used to denote different concepts. This difference is marked in WN by a different synset offset index as for instance in the typical case of PLANT, which is associated to the following four synsets:

```
00014510     plant, flora, plant_life
03806817     plant, works, industrial_plant
05562308     plant
09760967     plant
```

From a lexicographic point of view these four entries instantiate totally different senses and are associated with different glosses. As can be seen, the offset indices are very far from one another, thus indicating the distance in meaning involved in each of the different lemma forms. Now this is what we find with common nouns: it would be impossible to have a duplicate of the same lemma in adjacency within the same semantic lexical field indicating or instance a slightly different meaning. Polysemous words in WordNet are not many, and their presence in distant and different semantic lexical fields is an indication of the high frequency of usage of the word in the language.

The problem is that WordNet uses a similar technique to store information about "polysemous" proper names. In fact, this may sound quite strange, seeing that the only meaning associated to a proper name is the referent which they should designate. So what WordNet is actually highlighting by associating a synset to proper names is, on the one side, the possibility that two or more proper names share part of the name. This is usually the first name for person names and the name as identifier of different types of named entities, like a famous work of art, or a famous book, etc. As an example, here is the list of different entries associated to JOHN, first lemma:

```
06043175     John, Gospel_According_to_John
10364758     John, Saint_John, St_John, Saint_John_the_Apostle,
             St_John_the_Apostle, John_the_Evangelist,  John_the_Divine
10365110     John, King_John, John_Lackland
```

We have a first mention of JOHN as first member of a synset at 06043175, but then the two following mentions appear one adjacent to the other - thus belonging to the same semantic field (but is this a field at all?). On the other side, we know that when a person name is involved, then the title or the surname is usually needed to address the right person. This might also not be sufficient, but it is obvious that first names are totally ambiguous, without having to be regarded polysemous. Besides, we know that the concept is denoted by the full content of the synset, besides the gloss. And as the content makes it clear, we are here dealing with three totally different referents: one is the Gospel, the other is the Apostle and the third a King. So why use JOHN as first lemma and not the more distinctive second (or third if available) lemma? We find this to be totally misleading from a semantic point of view, because here we are not dealing with polysemous words as was the case with PLANT, but rather with referential identity. Besides, the word JOHN by itself can have additional uses. Consider for instance the corresponding lower case word "john" which is used with ambiguous meanings:

```
10076833     whoremaster, whoremonger, john
04274300     toilet, lavatory, lav, can, john, privy, bathroom
```

[4] In his lecture in 1970, and then published in the book Naming and Necessity, by Saul A. Kripke, 1980, Blackwell, Harvard University Press.

Here "john" is not the first member of the synset but the difference in meaning is clearly understood by a native speaker, and is testified again by the distance in terms of offset index values. In these two cases, the choice of lexicographers was not to highlight the polysemy of "john" which appears included in the set but not in first place, and will be assigned the corresponding offset index.

There are only sparse cases of first names as first lemmata in adjacent synsets before reaching the lexicographically marked section of the Noun file where all proper names are collected. Then, the choice to use first names as first members of the synset becomes very common in the more restricted list of person names made up of some 3200 entries that start around offset index 110102000. Here are some examples:

```
110102151    Aaron
110102325    Aaron, Henry_Louis_Aaron, Hank_Aaron

110103348    Adam, Robert_Adam
110103502    Adams, John_Adams, President_Adams, President_John_Adams
110103654    Adams, John_Quincy_Adams, President_Adams,
                   President_John_Quincy_Adams
110103839    Adams, Sam_Adams, Samuel_Adams

110105319    Agrippina, Agrippina_the_Elder
110105487    Agrippina, Agrippina_the_Younger

110109993    Allen, Ethan_Allen
110110169    Allen, Woody_Allen, Allen_Stewart_Konigsberg
110110327    Allen, Gracie_Allen, Grace_Ethel_Cecile_Rosalie_Allen, Gracie

110112423    Anderson, Carl_Anderson, Carl_David_Anderson
110112636    Anderson, Marian_Anderson
110112784    Anderson, Maxwell_Anderson
110112893    Anderson, Philip_Anderson, Philip_Warren_Anderson,
                   Phil_Anderson
110113110    Anderson, Sherwood_Anderson
```

and the list may continue. In order to cope with this uncouth and unmotivated choice, the algorithm had to be modified: now there would be uncertainty in both files. In the list of LFs, where more than one LF would be associated to each sense and would start with the same word. And in the gloss offset index + synset, where the same first lemma appearing in more than one synset, now was used to denote a different concept in adjacency. There was no way to use the same automatic approach we used previously. So in order to complete work on the NOUN data file, we have decided to disambiguate each and every synset that needed it: i.e. all those synsets that were associated with more than one LF. After manual modifications, here below is the output and the input for the sequence of adjacent "Anderson":

```
synset(110112423,anderson_NN(x1),['Anderson','Carl_Anderson','Carl_David_Anderson'])-
[united_NN(x1,e6),state_NN(x2,e5),physicist_NN(x3,e5),discover_VB-
[e1,x1,x4],antimatter_NN(x4),in_IN(x4,x5),form_NN(x5),of_IN(x5,x6),antielectron_NN(x6),
call_VB-[e3,x6,e3],positron_NN(x7,e3)]

synset(110112636,marian_anderson_NN(x1),['Marian_Anderson','Anderson'])-
[united_NN(x1,e2),state_NN(x2,e2),contralto_NN(x3,e2),note_VB-[e1,x1],for_IN-
[e1,x4],performance_NN(x4),of_IN(x4,x5),spiritual_NN(x5)]

synset(110112784,anderson_NN(x1),['Anderson','Maxwell_Anderson'])-
[united_NN(x1,e1),state_NN(x2,e1),dramatist_NN(x3,e1)]

synset(110112893,philip_anderson_NN(x1),['Philip_Anderson','Anderson','Philip_Warren_Anderson'
,'Phil_Anderson'])-[united_NN(x1,e2),state_NN(x2,e2),physicist_NN(x3,e2),study_VB-
[e1,x1,x4],electronic_JJ(x4),structure_NN(x4),of_IN(x4,x5),magnetic_JJ(x5),disordered_JJ(x5),s
ystem_NN(x5)]

synset(110113110,anderson_NN(x1),['Anderson','Sherwood_Anderson'])-
[united_NN(x1,e2),state_NN(x2,e2),author_NN(x3,e2),works_NN(x4,e2),be_VB-
[e1,x5,x4],frequently_RB(x5,e2),autobiographical_JJ(x5,e2)]
```

which was done after transforming the LFs as follows,

```
lf(anderson_NN(x1),[united_NN(x1),state_NN(x2),physicist_NN(x3),discover_VB(e1,x1,x4),antimatt
er_NN(x4),in_IN(x4,x5),form_NN(x5),of_IN(x5,x6),antielectron_NN(x6),be_VB(e2,x6,e3),call_VB(e3
,x8,x6),positron_NN(x7)]).
lf(marian_anderson_NN(x1),[united_NN(x1),state_NN(x2),contralto_NN(x3),note_VB(e1,x6,x1),for_I
N(e1,x4),performance_NN(x4),of_IN(x4,x5),spiritual_NN(x5)]).
```

```
lf(anderson_NN(x1),[united_NN(x1),state_NN(x2),dramatist_NN(x3)]).
lf(philip_anderson_NN(x1),[united_NN(x1),state_NN(x2),physicist_NN(x3),study_VB(e1,x1,x4),elec
tronic_JJ(x4),structure_NN(x4),of_IN(x4,x5),magnetic_JJ(x5),disordered_JJ(x5),system_NN(x5)]).
lf(anderson_NN(x1),[united_NN(x1),state_NN(x2),author_NN(x3),works_NN(x4),be_VB(e1,x4,x26),fre
quently_RB(x5),autobiographical_JJ(x5)]).
```

and the offset indices+synsets accordingly,

```
110112423,Anderson, Carl_Anderson, Carl_David_Anderson
110112636,Marian_Anderson, Anderson
110112784,Anderson, Maxwell_Anderson
110112893,Philip_Anderson, Anderson, Philip_Warren_Anderson, Phil_Anderson
110113110,Anderson, Sherwood_Anderson
```

4 Conclusion and Future Work

In this paper we presented ongoing work to produce a parser for Logical Forms resources that checks for their consistency, which is basically focussing on the existence of disconnected and ungrounded variables, and tries to correct them. This problem is divided up into two separate processes: one that looks for object variables and tries to connect them to the predicate they depend on. Another process looks for arity of arguments in any predicate formula in order to eliminate unwanted and unneeded variables: these may ensue basically due to the use of a basic lexical structure in presence, however, of omitted arguments. Arguments may be omitted either because they are optional, or because the predicate is used in a passive, intransitivized or ergativized construction. We found an amount of disconnected variables that averages 56% of all LFs, that is 71000 wrong entries over 138000 overall. After running the algorithm for correction which used a lexicon of 7000 verb entries, we managed to correct over 32% of LFs thus reducing the error rate to 24%. We worked then at manually correcting those LFs that are marked as inconsistent by the parser, overall some 4000 entries. We corrected in this way 5.64% of errors that were signaled by the parser. Intervening in this way we discovered new mistakes that are due simply to specific type of structures, containing adjunct structures at verb level. This will require a new effort to count these new mistakes and then manually check the remaining entries. We are also enriching semantically the logical forms, by two types of operations: signaling modifiers' semantic nature as being either restrictive or non-restrictive, then intersective, non-intersective and anti-intersective. But also treating three-place predicates distinguishing closed arguments from predicative arguments.

REFERENCES

Agerri, R. and Anselmo Peñas (2010) *On the Automatic Generation of Intermediate Logic Form for WordNet glosses*, 11th International Conference on Intelligent Text Processing and Computational Linguistics (Cicling-2010), LNCS Vol. 6008, Springer.

Alshawi, H., Pi-Chuan Chang, M. Ringgaard. (2011) Deterministic Statistical Mapping of Sentences to Underspecified Semantics, in Johan Bos and Stephen Pulman (editors), Proceedings of the 9th International Conference on Computational Semantics, IWCS,15-24.

Bos Johan & Rodolfo Delmonte (eds.), (2008) Semantics in Text Processing (STEP), Research in Computational Semantics, Vol.1, College Publications, London.

Branco, A. (2009) "LogicalFormBanks, the Next Generation of Semantically Annotated Corpora: key issues in construction methodology", In Klopotek, et al., (eds.), Recent Adavnces in Intelligent Information Systems, Warsaw, 3-11.

Bender, E.M. and D.Flickinger (2005) Rapid prototyping of scalable grammars: Towards modularity in extensions to a language-independent core. in Proc. 2nd IJCNLP-05, Jeju Island, Korea.

Bender, E.M., D.Flickinger, and S.Oepen (2002) The Grammar Matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In J.Carroll et al.(Eds.), Proc. Workshop Grammar Engineering and Evaluation at COLING19, Taipei, Taiwan, 8-14.

Cahill Aoife, Mairead Mccarthy, Michael Burke, Josef Van Genabith, Andy Way (2007) Deriving Quasi-Logical Forms From F-Structures For The Penn Treebank, in Studies in Linguistics and Philosophy, Vol. 83,pp 33-53.

Clark, Peter, Fellbaum, Christiane, and Hobbs, Jerry (2008) Using and Extending WordNet to Support Question Answering. In: Proceedings of the Fourth Global WordNet Conference, eds. A. Tanacs, D. Csendes, V. Vincze, C. Fellbaum and P. Vossen. University of Szeged, Hungary, pp. 111-119.

Copestake, Ann, (2009) Invited Talk: Slacker Semantics: Why Superficiality, Dependency and Avoidance of Commitment can be the Right Way to Go. In: Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), Athens, Greece, pp. 1-9.

Davidson, D. (1967) The logical form of action sentences. In N. Rescher (Ed.), The Logic of Decision and Action, Pittsburgh. University of Pittsburgh Press.

Davidson, D. (1980) Essays on actions and events. Oxford: Clarendon Press.

Harabagiu, S.M., Miller, G.A., Moldovan, D.I.: eXtended WordNet - A Morphologically and Semantically Enhanced Resource (2003) http://xwn.hlt.utdallas.edu, pp. 1-8.

Hobbs, J. (2005) Toward a useful notion of causality for lexical semantics. *Journal of Semantics,* 22:181–209.

Hobbs, J. (2008) Encoding commonsense knowledge. Technical report, USC/ISI. http://www.isi.edu/~hobbs/csk.html.

http1. http://wordnetcode.princeton.edu/standoff-files/wn30-lfs.zip.

http2. http://wordnetcode.princeton.edu/standoff-files/cwn-noun-lfs.txt.

Information Science Institute, University of Southern California: Logical Forms for WordNet 3.0 glosses (2007) http://wordnetcode.princeton.edu/standoff-files/wn30-lfs.zip

Mihalcea, R., and Dan I. Moldovan, (2001) *eXtended WordNet: progress report,* In: Proceedings of NAACL Workshop on WordNet and Other Lexical Resources, Pittsburgh, 95-100.

Moldovan, D. and V. Rus (2001) Explaining answers with extended wordnet. In Proc. ACL'01.

Rathod, Nishit & J. Hobbs (2005) Lftookit. In http://www. isi.edu/rathod /wne /LFToolkit/ index.html, 2005.

Rus, Vasile (2004) A first evaluation of logic form identification systems. In Senseval-3, Association for Computational Linguistics, pp. 37–40.

Schubert, L. and C.Hwang (1993) Episodic logic: A situational logic for NLP. In Peter Aczel, David Israel, Yasuhiro Katagiri, and Stanley Peters, (Eds.), Situation Theory and its Applications, vol.3:303-337.